1. □□□□□□□□□□□□□

□

Query Processing is the process of selecting the best method to be used in connection to a database request. Query processing system minimizes the cost of processing a set of queries by minimizing the processing cost of each query separately. A separate plan is generated and executed for each query. Queries involved in day to day database application may be of different types such as queries with multiple relations, queries with single relations and multiple operations, multiple queries with multiple relations. The main problem in multiple query processing is that if a query is decomposed into sub queries that require operations at separated and remote located databases, the sequence and sites of the databases are determined while performing the operations such that the communication cost and processing cost for processing the query is minimized. This problem is complicated by the fact that query processing not only depends on the operations of the query, but also on the parameter values associated with the query. The major tasks in multiple query processing in multi database system are common operation or expression identification and global execution plan construction. Each query can have several alternative evaluation plans, each with a different set of tasks. The objectives in the multiple query processing are to increase system throughput and decrease single query response time.

Since the problem is NP hard, it can be solved using genetic algorithm along with near optimal solution. The genetic algorithm is a heuristic solution used to solve intractable problems in database design. Genetic algorithms are implemented as a computer simulation in which a population of abstract representations called chromosomes of candidate solutions to an optimization problem evolves toward better solutions.

Individual plan in a query is represented as chromosome and individual task in a plan is represented as gene. Since a gene in a chromosome represents the

plan selected for the query corresponding to the gene position, in the mutation operation the plan number is only replaced with randomly selected valid plan's number for that query.

The main aim in this work has been focused to

(i) generate plans , cost of plans of multiple queries in heterogeneous query processing environment,

(ii) calculate average sum of the plans and degree coefficient between the tasks within the plans in heterogeneous query processing environment , as well as multi dependent query execution environment ,

(iii) avoid query processing problems due to the use of incorrect and partial information at optimization time, and

(iv) dynamically determine and assign different plans for different subsets of data.

□□□ □□□□□□□□□□□□□□□□□□□

The query size, number of relations, population size is assigned and the size of the chromosome is fixed along with the probability for crossover and probability for mutation. The initial population is built. The plan value is evaluated. The plan select value is evaluated by considering the size of the query and chromosome. The cost of plan is evaluated. The fitness value, weight of the plan, plan select value along with the average association degree coefficient in multiple heterogeneous query processing environment as well as multi dependent query processing environment are also evaluated. An optimal join strategy in relations is obtained from a structured query set along with the resulting plan value and functional value. Applying the hash functions to the relations, attribute join cost is calculated.

3. Problem Statement

In dealing with multiple heterogeneous queries as well as multi dependent queries, various parameters are required to be considered to evaluate the following.

- The size of query plans and tasks in multiple heterogeneous query processing environments.
- The cost of query plans, plan select value as well as degree coefficient between tasks within plans along with varying sizes of chromosomes.
- The weight of the plan, fitness parameter of query plans multiple heterogeneous query processing environments.
- The query plans and degree coefficient between query plans in multi dependent query processing environment.
- The cost of query plans along with the functional values by applying join enumeration techniques in the dynamic programming environment.
- The attribute join cost using join dependencies especially hash join implementation.

4. Problem formulation

4.1. Retrieving plan size, cost of plan along with fitness value, plan-select weight of the plan in multiple heterogeneous query processing and multiple dependent query processing environment

While keeping the size of chromosome =20, Pc (Probability for crossover operation)=0.02, Pm (Probability for mutation operation)=0.07 along with size of population =100 and the number of queries associated=100 , it is observed that the fitness parameter of the query plan is dependent to the size of the query. Similarly in multiple dependent query processing environment, while setting the size of chromosome to 30, probability for crossover operation to 0.09 and the probability for mutation to 0.04 it is observed that the query plan is dependent on the weight of the plan which can be obtained by obtaining the plan value and the size of the query.

Step 1 : - Submit the initial query and do the search

- Assign the query size, number of relations, population size and fix the size of the chromosome.
- Assign the value of Probability for crossover, Pc and the Probability for mutation, Pm
- Build the initial population

Step 2 : - For each sub-query of the population do the search

- build the local list of data sets End for

Step 3 : -Build a merged list

-Renew the sub-query

-Judge the top twenty data-set

-Take top twenty data

Step 4 : -For each sub-query N(s) of the population

-parent1= Selection (N(s))

-parent2= Selection (N(s))

-Crossover (Pc , parent1, parent2,son)

-Mutation (Pm , son, sonmut)

-Add_Sub-query (sonmut,N(s+1))

-Sub-query_size (N(s+1))=Sub-query_size (N(s))

-End for

Step 5 : -Evaluate the plan value

Step 6 : -Evaluate the plan-select value by considering the size of the query and chromosome

Step 7 : -Estimate the cost of the plan by evaluating plan select value number of query , number of relations and CPU time

Step 8 : -Evaluate the weight of the plan by considering the query size and plan value

Step 9 : -Compute the fitness of each individual plan in a query by estimating weight of the plan and query size

Step 10 : - Evaluate average association degree coefficient between plans by considering plan select value, number of relations and query size

□

□□□□□□□□ □□□□□□□□ □□□□

Murat Ali Bayir et.al[6] have formulated multiple query optimization by applying genetic algorithm. By using multiple query optimization problem, they optimized a set of queries together. Each query qi has a number of possible solution plans mi, and each plan of a query contains a set of tasks, which when executed in a certain order will produce the answer for the query. Each task has associated with cost value, which is a positive integer. They have also applied heuristic techniques to initiate the search with an initial state, which has all null values denoting that no plan has yet been selected for any queries.

Narasimhaiah Gorla et.al[9]have focused on Sub query allocations using genetic algorithms. In their work they mentioned that minimization of query execution time is an important performance objective in databases design. The problem of finding the minimum cost allocation is NP hard. In order to deal with their problem, GA is used to arrive at near optimal solution. GA is a heuristic solution that has been used to solve intractable problems in database design. The initial step of heterogeneous query processing is to transform a high level global query into an efficient execution strategy on local databases. The output of a sub query is an intermediate relation, which is stored after the query is answered.

M.Sinha et.al.[8] have also used genetic algorithms for query optimization. They have defined the query optimization as a complex search task, e.g. a search for the best possible plan from among the semantically equivalent plans that can be generated for any given query. It therefore seems logical to consider query optimization in terms of search algorithms. When genetic algorithm is applied to query optimization, the initial population is generated randomly. For each generation, genetic operations are carried out and thus the population evolves,

usually decreasing the average cost of its individuals. When the most optimal plan is obtained, it is passed to the database engine to execute the query. Optimization of queries can be done through two approaches, one consisting of algebra manipulations or transformations and the other including strategies to take advantage of the storage of the relations.

Cristina Lopez et.al [2] have evaluated the performance of different fitness functions generated from the plans based on the order of information retrieval. The population of individuals known as chromosomes, represent the possible solutions to the problem. These are randomly generated, although if there is some knowledge available concerning the said problem, it can be used to create part of the initial set of potential solutions.

T.Sellis et.al [11] have implemented a heuristic algorithm which performs a search over some state space defined over access plans. The search space is constructed by defining over state for each possible combination of plans among the queries.

Zehai Zhou et.al[13] have used heuristics and genetic algorithms for large scale database query optimization problem. Join methods are one of the major factors to be taken into consideration in modeling query processing. They have presented several new variants of the classical traveling salesman problem which leads to the cost estimation for query optimization.

In dynamic programming environment, while setting the size of chromosome to 9 along with the value of Pc=0.9 and the value of Pm=0.007, it is observed that the estimated cost of plan increases with the increasing value of resulting plan. Also the plan select value increases with the increasing value of resulting plan.

□□□□□□□ □□□□□□ □□

Input : A structured query set q1 to qn.

Output : An optimal join strategy in relations

Step-1 - for i=1 to size of query

create table access query plans of qi

Step-2 - for q=2 to size of query

for smallsize=1 to q/2

largesize=q-smallsize

Step-3 - for each smallquery of size smallsize

for each largequery of size largesize

if smallquery ∩ largequery != null then

continue

if not (smallquery is linked with largequery) then

continue

Step-4 - resulting_query_plan= (sizeof query*plan value)/relation;

Step-5 - planselect(i)=plan(i)/resulting_query_plan;

Step-6 - function value(i)=est_cost of plan(i)/plan value;

resulting plans = newly join plans

Step-7 - Activate plans by considering function value, resulting plans

and query size

return

4.2.2. Related work

Ismail H. Toroslu et.al[4] have formulated the solution on dynamic programming in multiple query optimization environment. They have defined the task as a database operation, which may partially or sometimes fully solve a database query, and it has an associated cost. Accordingly a plan is a set of tasks that can solve a database query. They have considered a number of plans to solve

the relevant queries. In the dynamic programming formulation of the multiple query optimization problems, all candidate solutions are generated to find a solution with the minimum cost. The dynamic programming algorithm stops generating candidate solutions when a candidate generated at certain cost is determined to be a solution.

Initially the Size of chromosome is set to 5 along with the following parameters,
Size Relation, R = 20 and for each tuple r in relation R
Size of Relation, S = 20 and for each tuple s in relation S
Size of query = 50
Probability of crossover (Pc) =0.07
Probability of mutation (Pm) =0.005

Step1 : - for (each tuple r in R relation)
apply hash function to the join attributes of r;

Step2 : - put r into the appropriate position in R;

Step 3 : - for (each tuple s in S relation)
apply hash function to the join attributes of s;

step 4 : - put r into the appropriate position in S;

Step 5 : - for (i=1;i<=n;i++)
build the hash table from R[i];
for (each tuple s in S[i])
apply hash function to the join attributes of s;

Step 6: - Estimate values to the hash functions for both the relations R and S

Step 7 : - Evaluate the attribute join cost by considering the hash functions to both the relations R and S

Usually a simple hash join is performed in two phases. In the building phase, the inner relation is hashed into main memory. The join attributes are used as a hash key. It may be assumed that a simple algorithm with no overflow handling requires that the entire inner relation fits in main memory. After this, the second phase called probing phase is performed. In the probing phase, the outer relation is read sequentially and for each record in the outer relation the matching records in the inner relation are retrieved.

Wei Li et.al.[12] have focused on the partition skew which is of concern in hash-based join. In the first step of hash join, some buckets may contain more tuples than other buckets due to an interaction between the distribution of attribute values and the hashing function itself. When this disparity becomes large, the bucket no longer fits in main memory and hash-based join degrades into nested-loop join.

Chun Jin et.al[1] have focused on incremental evaluation on SJP (Selection – Join- Projection) queries and query optimization. They have conducted the experiments on two query sets. The query network for each individual query was generated and evaluated. While running the queries on the specific conditions, the query networks with transitivity inference gain significant improvements over directly running the SQL queries on the DBMS.

M. Kitsuregawa et.al.[7] first introduced the GRACE hash join algorithm . Many refinements of the algorithm have already been proposed for the sake of avoiding I/O by keeping as many intermediate partitions in memory as possible.

J. Gray et.al[5] have discussed in their paper to handle Skewed Data. While the uniform distribution offers the best case for parallel scalability, skewed input distribution would test the ability of the parallel hash join implementation to handle load imbalance. The serial performance of their hash-join algorithm for skewed data was comparable to the serial performance for uniform data. When all partitions are not uniformly sized, the amount of time to join each partition varies. In the extreme case when all the tuples fall into a single partition, no parallel scalability were possible with the naive implementation.

H. Zeller et.al.[3] while using their hash join algorithm share two common building blocks, (1) partitioning , and (2) joining with in-memory hash tables. To

cleanly separate these two phases, they have used GRACE as their baseline algorithm throughout the evaluation.

Shimin Chen Anastassia et.al.[10]have focused on improvement of Hash Join Performance through Pre- fetching. Hash join is commonly used in commercial database systems to implement equijoins efficiently. The Hash join algorithm first builds a hash table on the smaller relation, and then probes this hash table using tuples of the larger relation to find matches. However, the random access patterns inherent in the hashing operation have little spatial or temporal locality.

5 [illegible]

[illegible]5 [illegible]

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
1	783650	0.13262	0.5	0.5	100.18
2	853602	0.13612	0.5	0.5	100.18
3	1013865	0.14413	0.5	0.7	100.3
4	2101057	0.15394	0.5	1.6667	100.4
5	316712	0.10927	0.5	0.15286	100.08

It is observed that, the cost of the query plans very much depend on the size of query plans. As the size of query plan increases, the estimated cost of query plan is also increased. The plan select value increases with the high range of query plans. The plan select value can be obtained by evaluating plan size along with the query size. It has also been seen that the fitness parameter of a query plan is dependent on the weight of the plan as well as the size of the query. The weight of a plan can be obtained by evaluating the plan value and the size of the query.[illegible]

Table 5.2. Resulting Plan value along with the Cost of Plan, Plan Select value and Functional value

Sl.No.	Resulting Plan value	Est_Cost of Plan	Largesize (size of access plan)	PlanSelect value	Functional value
01	237	0.0042133	57	10.533	0.00046815
02	194	0.0034489	48	8.6222	0.00038321
03	447	0.0079467	67	19.867	0.00088296
04	354	0.0062933	63	15.733	0.00069926
05	164	0.0029156	47	7.2889	0.00032395

It is seen that the plan select value is directly proportional to the functional value. It has also been observed that the functional value increases with the increasing size of the resulting plan value. The size of access plan is quite dependent upon the resulting plan value. The cost of query plan increases with the increasing size of resulting plan value.

Table – 5.3. Hash function to the relations with attribute join cost

Sl.No.	H(r)	H(s)	Attribute Join cost
1	25	10	0.20444
2	24	9	0.19556
3	18	7	0.14489
4	15	6	0.12622
5	40	16	0.32622

While evaluating the attribute join cost using join dependencies, especially hash join, it is observed that the join dependencies impacts on the attribute join cost with the increasing functional values achieved from the relations. It is obvious that the hash join utilizes two hash table file structures to partition each relation's records into sets containing identical hash values on the join attributes.

□ □□□□□□□□□□

The query processing is an inherent activity for every database. In query processing scenario, long evaluation period may suffer from resource availability that may be changed during the long evaluation period of execution plans. As a result, queries are often evaluated with sub-optimal plan configurations. To provide solution to this situation, multiple query processing techniques are implemented. Once optimal execution plan is selected for a query then there is no need to create any execution plan for similar query in future.

The process of optimizing very large join queries is an extremely difficult combinatorial optimization problem. The computational experiments on several algorithms have been experimented using genetic algorithm. These computational results are generally more profitable to experiment with the genetic algorithms. While much research has been conducted to find the best set of parameters for a genetic algorithm to solve a particular problem, it seems that it would be difficult to find a set of parameters that are suitable for all instances for that particular problem.

Heuristics are not very expensive in terms of computation time. So it is also feasible to combine them with some other local improvement procedures in order to find better, local optimal solution. Second, since genetic algorithm randomly searches some point in the solution space, it may end up with a point which is even not the local optimal point. Therefore it is suggested that for query processing problems in general and for query optimization in particular, a post processing procedure may be added to the genetic algorithm. More specifically, some local improvement procedures may be suggested to be used to find the local optimal points after the genetic algorithm procedure is terminated. □

□ □ □ □

□ □□□□□□□□□□

[1] Chun Jin, "Optimizing Multiple Continuous Queries", CMU-LTI-06-009 Language Technologies Institute , School of Computer Science , Carnegie Mellon University 5000 Forbes Ave. Pittsburgh, PA 15213, 2006.

[2] Cristina Lopez, Vicente P.Guemero Bote , University of Extremadura , Badajoz, Spain , "Proc. Of ACM Sigmod Intl.conf.of Management of Data , June 2005.

[3] H. Zeller and J. Gray. An Adaptive Hash Join Algorithm for Multiuser Environments. In the 16th VLDB, pages 186.197,Aug. 1990.

[4] Ismail H. Toroslu , Ahmet Cosar , "Dynamic programming solution for multiple query optimization problem", Information Processing Letters 92 (2004) 149–155 , Science Direct , Elsevier.

[5] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. In SIGMOD Conference, pages 243–252, 1994.

[6] Murat Ali Bayir, Ismail H.Toroslu , and Ahmet Cosar , "Genetic algorithm for the multiple query optimization problem", IEEE transactions on Systems , Vol-37, No.1, January 2007.

[7] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka. Application of Hash to Data Base Machine and its Architecture. New Generation Computing, 1(1):63.74, 1983.

[8] M.Sinha , S.V. Chande□, "Query optimization using Genetic algorithm" , Research Journal of Information Technology , pp-139-144 , 2010.

[9] Narasimhaiah Gorla, Suk-kyu Song, "Subquery allocations in distributed databases using Genetic algorithms", JCS&T Vol-10, No.1, April-2010

[10] Shimin Chen Anastassia Ailamaki Phillip B. Gibbons, Todd C. Mowry, " Improving Hash Join Performance through Prefetching", Carnegie Mellon University , Intel Research Pittsburgh, 2003.

[11] T.Sellis, "Multiplequery optimization",IEEE transactions on knowledge and data Engineering , Vol-2, June-1990.

[12] Wei Li, Dengfeng Gao Richard T. Snodgrass , "Skew Handling Techniques in Sort-Merge Join Oracle Corporation" , ACM SIGMOD '2002 June 4-6, Madison, Wisconsin, USA , Copyright 2002 .

[13] Zehai Zhou, Journal of Information of Computing Science, Vol-2, No.4, 2007.